STA 32 R Handout 7, Miscellaneous Things in R

This handout is to go over different things that one should know how to do in R.

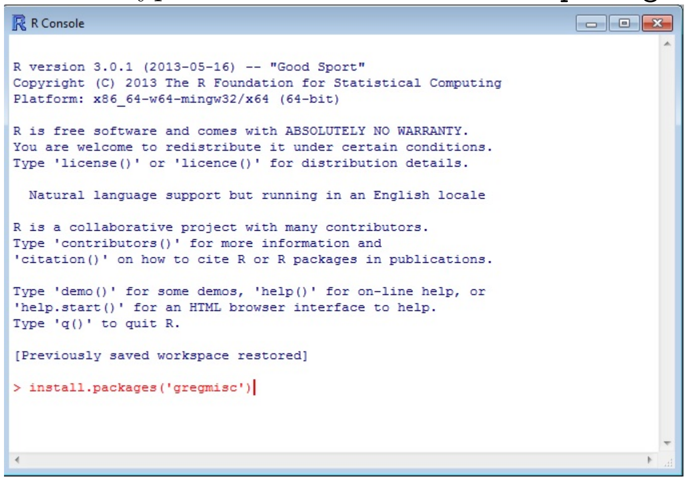
1 Packages and Libraries

Just like in many other programming languages, there is no one package/library that does everything. In R, for example, there are many optimization functions that people have written and put together into a package, but they are usually not installed in the base R. We will need to search for that package and install it onto our local machine before we can use those functions. The basic procedure for installing and using a new package is as followed:

1. Make sure your computer is connected to the internet
2. Use the command `install.packages('the package name')` (with quotes) to install the package
3. Use the command `library(the package name)` (without quotes) to load in the library.

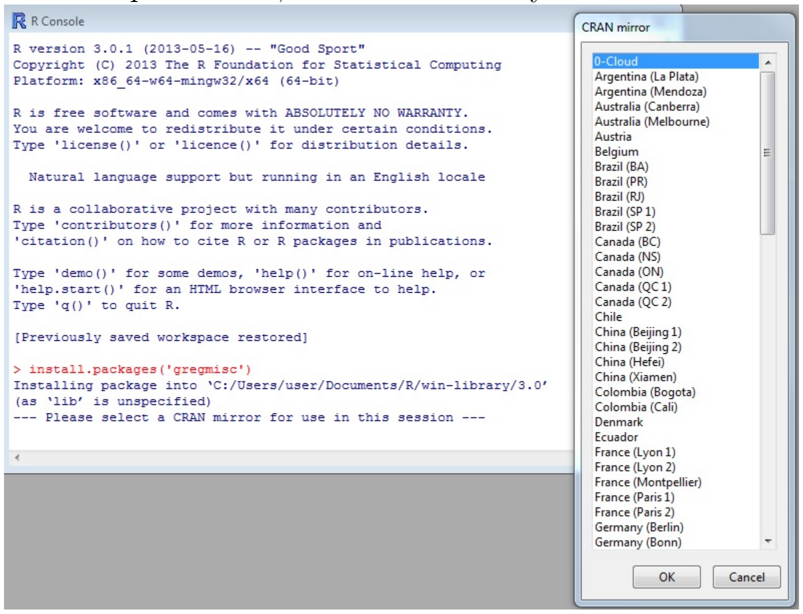
Suppose we want to create all combinations of a set of numbers. Surprisingly there is no easy way to do this in the base `R`. Nice enough, there is a the function `combinations` in the package `gregmisc` that we can use. Here are some screenshot procedures using the `R` GUI:

1. First we type in the command `install.packages('gregmisc')`



```
R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> install.packages('gregmisc')
```

2. Once we press enter, a window that says CRAN mirror will pop up



```
R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> install.packages('gregmisc')
Installing package into 'C:/Users/user/Documents/R/win-library/3.0'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
```

That is for you to choose which location server you want to install your package from. If you scroll down you should see USA (CA 1), which is located at Berkeley, and USA (CA 2), which is located at LA. Choose either one and click ok. (Note: Sometimes there might be an error install from one of these servers, if that happens, install your package from a different server usually solves the problem.)

3. Depending on the package size and your internet speed, it can take a couple seconds up to a couple minutes to install the package. Once its done, just type in `library(gregmisc)` to load in the package. Now you can start to use the functions inside this package.

4. Now suppose we want to create all possible combinations when we draw 3 items from the number list `1:4`. Recall this gives us $\binom{4}{3} = 4$ ways. The command to do this is `combinations(n = 4, r = 3, v = 1:4)`, where n is the length of the source, r is the length of the target (i.e. how many items we want), and v is our source.

```
combinations(n = 4,r = 3, v = 1:4)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    4
[3,]    1    3    4
[4,]    2    3    4

combinations(n = 4,r = 3, v = c("a","b","c","d"))
     [,1] [,2] [,3]
[1,] "a"  "b"  "c"
[2,] "a"  "b"  "d"
[3,] "a"  "c"  "d"
[4,] "b"  "c"  "d"
```

Remarks:

- Once you have installed a package, you don't need to install it anymore (unless you reinstall R or upgrade R)
- You need to load the package you want to use everytime you start a new R session.

2 Plotting

We have seen how to do histogram and boxplot in R. We will now learn a few more functions to do plots.

2.1 plot()

The most basic plotting function is the `plot()` function. It takes in a lot of argument, but the most important argument is, of course, the data. Suppose you want to plot pairs of data points. You can supply both the x-coordinates and y-coordinates, or you can just supply the y-coordinates, and R will treat the x-coordinates as 1,2,3,... up to the number of data points you have.

```
x = rpois(10, lambda = 4)  #Randomly generate 10 x-coordinate
y = rpois(10, lambda = 4)  #Randomly generate 10 y-coordinate
plot(x,y)   #The first argument is the x-coord., and second argument is y-coord.
plot(y)     #The x-coord. is now 1:10.
```

This will only plot points. We can use different argument to create different types of plots.

```
plot(x,y,type = 'l')  #Plot lines
plot(x,y,type = 'l', lty = 2) #The lty argument gives different line type
plot(x,y,type = 'h')  #Plot spikes
```

Suppose you want to add another data set onto the graph. There are a couple ways to do that, depends on the type of graph you want to add on.

```
plot(x, y, xlim = c(0,10), ylim = c(0,10))   #Plot the original data,
# and set x and y plotting range
newx = 1:10; newy = 1:10   #Create some new data point
lines(newx, newy)          #Add a diagonal straight line onto the plot
points(newx, newy, col = 'red', pch = 23)    #Add the new data as points,
# in red and diamond shape
plot(0:25, pch = c(0:25))   #Plots out all possible point style
```

To use the `line()` and `points()` functions, we must already have created a base plot, or else we can just use the arguments `type = 'l'` or `type = 'p'` in the `plot()` function to do these tasks. Also, everytime you use the `plot()` function, the new plot will overwrite the old plot.

2.2 Other useful plotting functions

- `curve()`: graph function of x. If argument **add** = **TRUE**, then it will draw graph on the current figure using current range of x values. By default, **add** = **FALSE**.

```
curve(dnorm, from = -5, to = 5, mu = 3)  #Graph the standard normal distribution.
curve(x^3 - 3*x, -2, 2)
curve(x^2 - 2, add = TRUE, col = "violet")    #Add the curve to the existing plot.
```

- `abline()`: the use of this function requires an existing plot. The `abline()` function adds lines to a figure. The arguments a, b are the intercept and slope of the line, h is used to plot a horizontal line, while v is used to plot a vertical line.

```
plot(x, y, xlim = c(0,10), ylim = c(0,10))   #Using the same x and y from above
abline(a = 2, b = 0.5, h = 2, v = 4, lty = 1:3) #Add a line y = 2+0.5x, a
#horizontal line y = 2, and a vertical line x = 4, with line type 1:3 respectively
```

- `rug()`: the use of this function requires an existing plot. This function adds lines along the x or y axis to show data values. By default, lines are drawn on the x-axis, use `side = 2` to draw on the y-axis.

```
plot(x, y, xlim = c(0,10), ylim = c(0,10))   #Using the same x and y from above
rug(x)
rug(y, side = 2)
```

- `arrows()`: the use of this function requires an existing plot. This function adds arrows to the plot. You need to specific the x,y coordinates of the arrow's tail (starting point), and at least 1 of the x,y coordinates of the arrow's head (ending point).

```
plot(x, y, xlim = c(0,10), ylim = c(0,10))   #Using the same x and y from above
arrows(4, 5, 6, 2)
```

- `text()`: this function adds text to an existing. You need to provide the location of the text (x and y coordinates). The argument **adj** tells `R` where to start the text: a value of 0 means the text starts at the provided location, while a value of 1 means the text ends at the provided location.

```
plot(x, y, xlim = c(0,10), ylim = c(0,10))   #Using the same x and y from above
text(6, 8, labels = "HELLO", adj = 0)
```

- `legend()`: use this to add a legend to the existing graph.

3 Checking Normality of a Data Set

We have learned the Normal Distribution in lecture. We often make assumptions that a given data set is normally distributed, so we can apply many existing techniques to analyze the data. However, it's always good to check whether our data is normally distributed first before apply any techniques that require this assumption. To assess normality graphically, we use the functions `qqnorm()` and `qqline()`.

```
x.norm = rnorm(100) #Generate 100 standard normal random variable
x.unif = runif(100) #Generate 100 uniform random variable
qqnorm(x.norm); qqline(x.norm)
qqnorm(x.unif); qqline(x.unif)
```

The function `qqnorm()` plots out the **normal probability plot** (see Sec 4.10 in the book for more details). The function `qqline()` draws a straight line across the points. Basically, if the data is normally distributed, the most of the points will lie approximately well on the straight line. We can see that for the `x.norm` data, most of the points lie on the straight line, where for the `x.unif` data, most of the points are away from the line.